SUPLEMENTO MENSUAL DEDICADO A LA IMAGEN SINTÉTICA

Rendermanía

Número 10

Cómo...
Renderización de archivos Dem

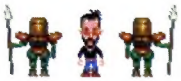
Taller Virtual
Luces flare para POV 3.0

Biblioteca 2D
Texturas planetarias

Foro del lector
Trabajos en 3D Studio

pcmanía

Todo los ficheros de inclusión y los ejemplos podéis encontrarlos en el directorio PCMANIA\RENDER60 del CD-ROM.



José Manuel Muñoz

Elevar ficheros Dem con POV

Por estas páginas ya han pasado diversas utilidades de alzado de terrenos (la última fue Hf-lab). Todas estas herramientas producen objetos de apariencia montañosa utilizando técnicas fractales. Pero, ¿qué hacer si uno está interesado en incluir zonas "reales" en una escena? La respuesta a esto son los ficheros .Dem (Digital Elevation Model), que guardan datos de alturas de zonas terrestres reales y pueden –mediante utilidades– ser "traducidos" a archivos tga procesables por «POV».

El usuario puede encontrar ficheros .dem en muchos sitios, pero sin duda el más importante es el Web del USGS (United States Geological Survey); un organismo a través del cual pueden conseguirse más de 700.000 fotos aéreas de nuestro planeta. Podemos comenzar en: **http://info.er.usgs.gov**

...y podemos buscar mapas en: **ftp://edcftp.cr.usgs.gov/pub/data**

Otro "site" muy útil es: **www.ems.uwplatt.edu/ce/fac/dymond/giswww.htm**

En este último lugar encontraremos una colección de sitios Gis recopilados por un tal R. Diamond. (las siglas Gis corresponden a "Geographic Information System" y aluden a un sistema de información diseñado para tratar informáticamente coordenadas geográficas. Sin embargo, en el faq gis-faq.txt –que incluimos en el CD-ROM– se indican un montón de definiciones más de las que parece desprenderse, que GIS es tanto una serie de bases de datos como un conjunto de procedimientos y normativas para procesar estos datos).

Pero volviendo al Usgs, este organismo tiene archivos .dem que cubren la totalidad del continente americano con una resolución de un valor de altura para cada 30 metros. Pero partiendo de estas direcciones, podemos hallar ficheros dem que cubren todas las zonas planetarias, información geológica, utilidades relacionadas con los datos ofrecidos, bitmaps de otros planetas del sistema solar y muchas cosas más.

En cuanto a la palabra dem, es empleada en muchos sitios como un término genérico que alude a todos los tipos de ficheros donde se almacenan datos de elevaciones de zonas geográficas. Sin embargo, aquí lo emplearemos para referenciar a los ficheros de formato dem distribuidos por el Usgs.

Los archivos dem están en formato ascii y son arrays de datos donde se guardan valores de altitud para distancias espaciales regulares de zonas geográficas reales. (es decir, los satélites aplican una rejilla imaginaria sobre el terreno y toman un valor de altura para cada intersección de esta rejilla). La pregunta ahora es: ¿cómo podemos emplear un archivo .dem desde POV?

Dem2pov

Dem2pov es una utilidad escrita por Bill Kirby partiendo de otra utilidad de conversión más general. Dem2pov lee ficheros .dem y produce archivos en formato tga que podrán ser leídos y empleados por la sentencia Heightfield de Pov. El programa funciona bajo DOS y requiere al menos un 386. La invocación más corriente para este programa es:

dem2pov fichero_entrada.dem fichero_salida.tga

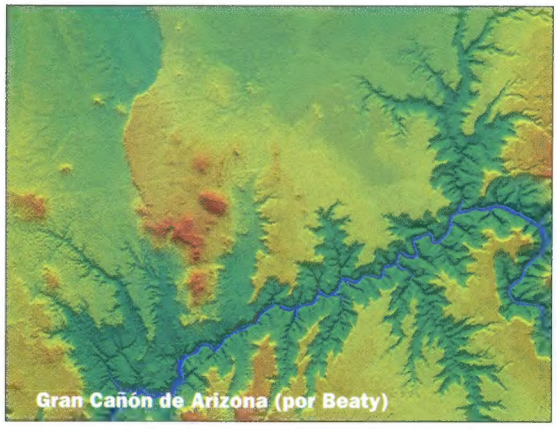

HFTEST3


HFTEST4

Sin embargo, antes de generar el fichero tga que emplearemos más tarde desde POV, Dem2pov nos hará una serie de preguntas cuyas respuestas afectarán al archivo a producir. Para estas respuestas podremos apoyarnos en una serie de datos que forman parte de la cabecera del .dem y que Dem2pov imprimirá en el monitor antes de formular las preguntas. También, si lo deseamos, podremos ordenarle al programa que se limite a mostrarnos dicha cabecera, sin generar ninguna tga (para ello le indicaremos tan sólo el fichero .dem de entrada). Además, es posible guardar esta cabecera en un archivo con una operación de redirección de ficheros como:

dem2pov fichero_entrada.dem > fichero_datos.txt

Pero regresemos al caso más típico. Después de mostrarnos los datos de la cabecera, el programa nos mostrará el mensaje "0 for all, 1 for samples, 2 for subset". Con ello el programa nos ofrece tres alternativas: si respondemos "0", Dem2pov entenderá que deseamos emplear todos los valores de alturas del dem. (Este es el caso más frecuente). Con "1" el programa creará una tga "a escala" tomando sólo uno de cada x valores de altitud. Este escalado no tiene por qué ser proporcionado ya


Gran Cañón de Arizona (por Beaty)

que, si escogemos esta opción, después se nos pedirá que introduzcamos el "colum sample interval" y el "row sample interval". Es decir, los intervalos para las columnas y las filas. Si damos un valor de 5 para ambos casos, por ejemplo, el programa sólo tendrá en cuenta un valor de altura de cada 5, lo cual quiere decir que la tga resultante será 25 veces más pequeña. Finalmente, el caso 2 se emplea para crear una tga usando sólo una parte del dem y desechando el resto de los datos de altitud. Aquí se hace preciso explicar que los datos de alturas del dem se ordenan en columnas y filas, empezando la primera columna en el borde oeste del mapa y la última en el lado este. Dentro de cada columna los datos empiezan en el Sur del mapa y acaban en el Norte. Sabiendo esto podremos especificar el área del dem a partir de la cual va a crearse la tga dando valores a las siguientes líneas; "start column" (primera columna), "end column" (última columna), "start row" (primera fila) y "end row" (última fila).

En la siguiente pregunta el programa nos pedirá que indiquemos el tipo de heightfield a crear. Los valores aceptables son 0 para los "Actual heights" y 1 para los "Normalized". Si nos decidimos por el primer caso, el programa nos pedirá después (con "vertical scaling factor") un valor que será empleado como factor de conversión entre unidades de medida (por ejemplo usaríamos 3.281 para pasar de metros a pies). En el otro caso, la cuestión es algo más complicada: la cabecera del Dem devuelve dos valores que son los valores de mínima y máxima altitud del fichero. Si escogemos el heightfield "normalizado" el valor máximo de altitud quedará a la altura 1 en el heightfield de POV (si no se hacen "scales"). El resto de los puntos a calcular tomarán sus valores de altitud teniendo en cuenta el siguiente factor:

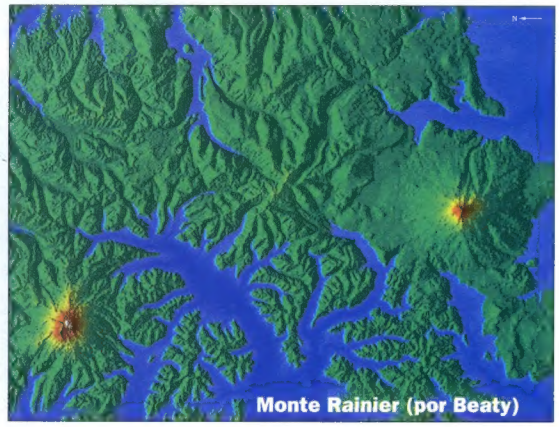
factor= 65535 / (Elevación_maxima + Bias)

El valor 65535 es el número de gradaciones de altura posibles que permiten los heightfields que emplean tgas. La variable "Bias" será la siguiente que daremos por teclado. Se trata de un valor que se sumará o restará a los valores de altitud dependiendo de su signo. Para comprender cómo funciona esto consideremos los siguientes ejemplos tomados del fichero Kirkland.dem (que se incluye con la utilidad): el valor de altitud mínima de este fichero es de 1158 metros y el de la máxima de 1820 metros. Si generamos la tga escogiendo la opción de normalización y dando un valor de 0 para la escala y el Bias, el valor de altura para el punto más bajo será de 0.6326 y de 1.0 para el punto más alto. Si, en vez de esto, damos un valor de -1158 al bias, entonces los resultados serán de 0 y de 1 respectivamente para la altitud de los puntos más bajo y más alto. Esto quiere decir, por supuesto, que, como podemos ver en las fotos hftest3 y hftest4 (página anterior), la forma final del objeto se verá afectada. La base del objeto queda en el mismo sitio para ambas pruebas pero, en el caso del bias=-1158, parece que el objeto se ha estirado hacia abajo.

Finalmente, el programa nos pedirá el valor de "default elevation (final output units)". Nosotros no acabamos de comprender la función de este parámetro. Según el doctorial se emplea para dar valores de altitud a los puntos del Heightfield para los que no haya valores dem (¿?). En las pruebas se ha dejado siempre a 0.

Los ficheros de ejemplo

Al descomprimir Dem2pov.zip, nos encontraremos el ejecutable de la utilidad, el código fuente (dem2pov.c), va-


Monte Rainier (por Beaty)

rios ficheros utilizados para compilar dichos fuentes en distintos compiladores, un fichero .pov para hacer pruebas (hftest.pov) y un zip con un archivo dem de ejemplo (kirland.zip). Como veremos, una de las líneas del fichero .pov escala el objeto heightfield que se creará. Este detalle tendrá la virtud de dar distintas proporciones al objeto, con lo cual quedará alterado el propósito inicial de renderizar un trozo de terreno real. ¿Por qué? Si los lectores quieren generar terrenos iguales a los almacenados en los archivos dem, la escala tendrá que ser uniforme en los tres ejes (Nota: estamos dando por sentado que conocéis perfectamente el funcionamiento correspondiente a la sentencia Height_field de POV; si este no es el caso puede hallarse una descripción de la misma en el artículo "Cómo crear terrenos..." del número 8 de Rendermanía).

El problema de la textura

Quizá el problema más importante que presenta cualquier Heightfield es el de la textura a aplicar. Si nos limitamos a dar un color liso al objeto, este no tendrá una apariencia demasiado real. Las montañas del planeta tierra no suelen exhibir un color uniforme. Pero, afortunadamente, la realidad puede simularse aplicando un mapa de colores sobre el objeto (como recordarán los povmaníacos, la sentencia color_map nos permitirá definir una gradación suave de colores sobre un objeto). Sin embargo, esto no será fácil. Primero habrá que definir un buen mapa de colores y luego habrá que aplicarlo adecuadamente (podemos ver lo que sucede si el objeto está desplazado o incorrectamente escalado con respecto al mapa de color mirando las imágenes hftest3 y hftest4).

Área de Leadville (por Beaty)

Las montañas de Bryan Beaty

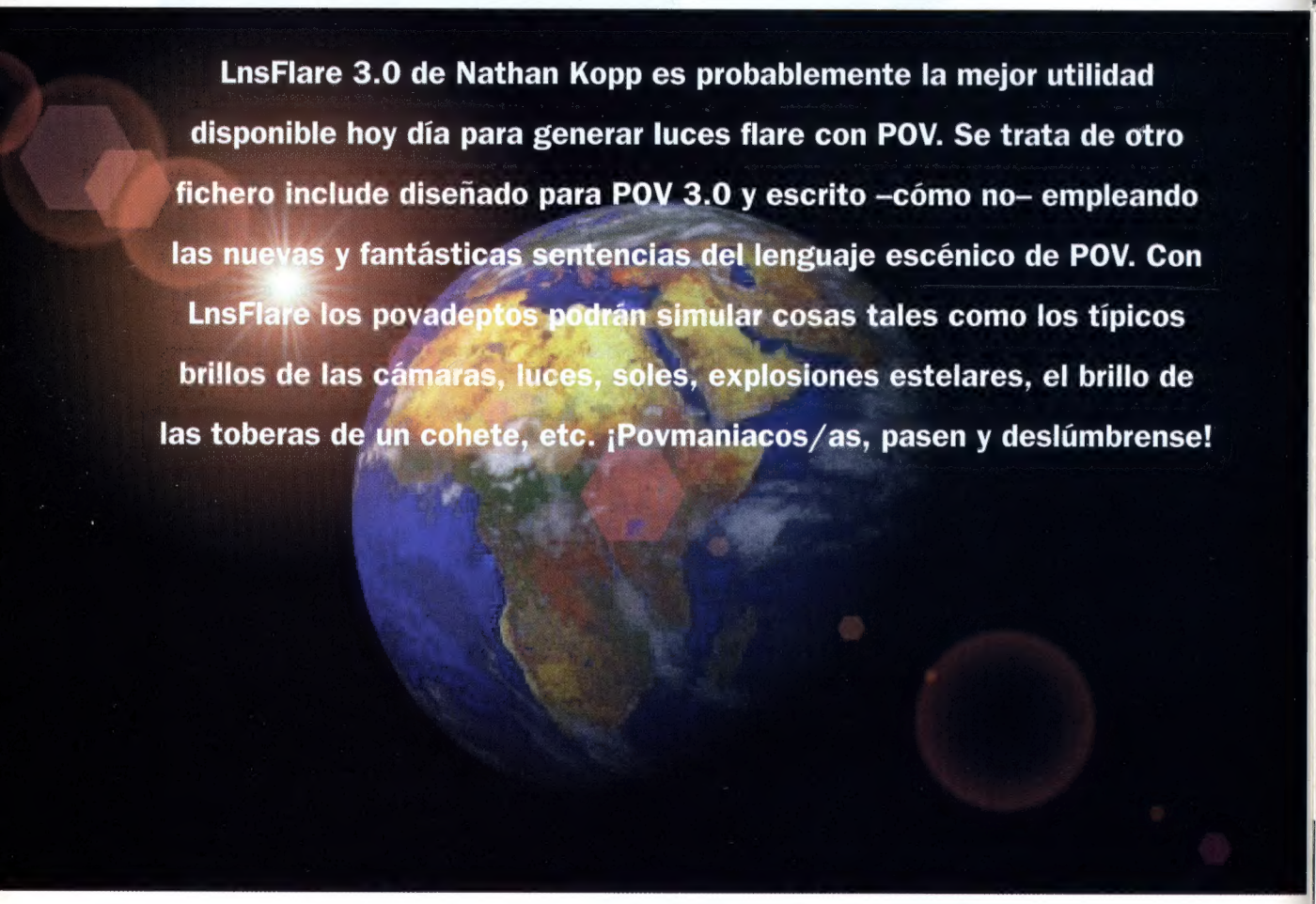
Bryan Beaty es un pov-usuario que suele utilizar ficheros dem para generar estupendas imágenes sintéticas. Hay que destacar, sin embargo, que una buena parte de ellas no están hechas con POV, sino con utilidades creadas por el propio Beaty. En efecto, este autor ha creado una serie de utilidades para manipular los archivos dem de diversas maneras. Según afirma el mismo Beaty, el primer paso consiste en traducir el dem a emplear a un formato más compacto y fácil de manipular (los archivos dem están en formato ascii y pueden ser muy grandes. Además, estos ficheros pueden tener muchas posibles cabeceras y especificaciones que hacen bastante fastidiosa y complicada la tarea de escribir utilidades para ellos. Hemos visto muy pocas utilidades para este tipo de archivos).

Por estas razones Beaty comienza empleando una utilidad escrita en C++ por él mismo y que genera un fichero en formato Tpatch a partir del Dem original. Este fichero Tpatch utiliza un formato inventado por el mismo Beaty que codifica cada dato de altitud en 2 bytes (lo cual representa un importante ahorro en el tamaño del fichero).

En el siguiente paso Beaty emplea otra utilidad que lee el archivo tpatch generado previamente y dibuja una imagen del terreno empleando un algoritmo propio. Las imágenes son vistas aéreas muy similares a las que podemos ver en los mapas físicos de cualquier Atlas. Según afirma el autor, esta utilidad genera la imagen empleando tres operaciones fundamentales: la interpolación, la codificación del color y el sombreado. En el primer paso, la interpolación puede ser necesaria si la imagen final a generar tiene una resolución superior a la del dem empleado como punto de partida. En este caso la utilidad generará puntos adicionales utilizando la interpolación entre los puntos originales. Después viene la codificación del color. Esto viene a ser algo equivalente al mapa de colores de POV. Los picos de las montañas tienen un color blanco, las zonas algo más bajas del mapa se colorearán con tonos marrones y amarillos, y las tierras bajas con colores verdes. Finalmente, en el sombreado, se utiliza la inclinación de las normales de cada triángulo para determinar la cantidad de luz que recibirá dicha cara con respecto a una fuente de luz situada de forma arbitraria al norte del terreno.

Para Beaty, sus imágenes tienen más de arte que de mapas, lo cual queda de manifiesto, sobre todo en la imagen que ha servido de portada a este artículo y que Beaty creó empleando POV (el site de Bryan Beaty está en www.oas.omron.com/bryan/anims.html).

Lnsflare.inc:



LnsFlare 3.0 de Nathan Kopp es probablemente la mejor utilidad disponible hoy día para generar luces flare con POV. Se trata de otro fichero include diseñado para POV 3.0 y escrito –cómo no– empleando las nuevas y fantásticas sentencias del lenguaje escénico de POV. Con LnsFlare los povadeptos podrán simular cosas tales como los típicos brillos de las cámaras, luces, soles, explosiones estelares, el brillo de las toberas de un cohete, etc. ¡Povmaniacos/as, pasen y deslúmbrense!

Quien haya leído los artículos anteriores sobre Trees.inc y Books.inc se hará una idea de cómo se trabaja con este nuevo "Ipas" de POV. Como ocurre con los mencionados ficheros, Lnsflare se maneja asignando valores a diversas variables para, finalmente, colocar una orden "#include" que cargará el flare en nuestra escena. La cantidad de parámetros que se requiere para definir un flare es bastante grande pero LnsFlare trae ya definidos 20 tipos de flares distintos. Con lo que bastará con declarar algunas variables e indicar el tipo de flare para colocar el "objeto" en nuestra escena.

Nuestro primer flare

Para situar un flare en una escena hemos de seguir una serie de pasos. En primer lugar hay que indicar a LnsFlare dónde se encuentra la cámara, lo cual se hará asignando a la variable cam_loc la misma posición que va a tener la cámara. De hecho, lo más práctico es hacer algo como esto...

Luces flare para POV

```
...
...
#declare cam_loc = <x,y,z>
camera{
  location cam_loc
  ...
  ...
```

Obrar así nos resultará lo más cómodo, ya que sólo tendremos que cambiar un vector cuando vayamos a cambiar la posición de la cámara. Igualmente deberemos obrar con la variable lookat, con la que indicaremos la posición hacia la que está mirando la cámara (vector look_at), y con la variable light_loc, en la que señalaremos la posición de la fuente de luz que va a ser "flareada". También hay que indicar el vector donde queda el cielo (para la cámara) en el identificador "sky_vect". Esto, en una escena típica donde el suelo corresponde al plano X-Z deberá hacerse con "sky_vect=y".

Y por último, tan sólo nos restará indicar el tipo de flare para la escena y colocar la sentencia include. Veamos un ejemplo;

```
...
...
#declare flare_type = "35mm"
#include "lnsflare.inc"
...
...
```

El array "35mm" asignado a la variable "flare_type" es el nombre de uno de los flares que vienen definidos en LnsFlare.

Experimentos

En tierra0.pov hay definido un flare del tipo "105mm" situado a 122.000 unidades de la cámara. La imagen resultante es la de una especie de estrella azulada de la que parten varios rayos de luz. Nuestro primer experimento consistirá en alejar este pequeño sol sumando un par de ceros a la variable "light_loc". El resultado de renderizar esto, sin embargo, no ofrecerá cambios con respecto a la imagen anterior. Esto ocurre así porque los flares son, en realidad, discos sobre los que se aplica una textura con la "luz". El flare, en la imagen, parecerá estar en la misma posición 3D de la fuente luminosa cuya ubicación hemos dado en "light_loc", pero pronto comprenderemos que realmente no está en el mismo sitio ya que, sin importar cuánto alejemos a la fuente, el flare seguirá teniendo el mismo tamaño.

Esto nos vendrá muy bien porque podremos dedicarnos a manipular el aspecto del flare sin que importe su tamaño real ni la distancia a la que se encuentre. Tenemos que pensar en el flare como un efecto que se sitúa sobre la fuente de luz, no como un objeto normal (es de suponer que la posición final del flare se calcula automáticamente a partir de la posición 2D de la fuente en

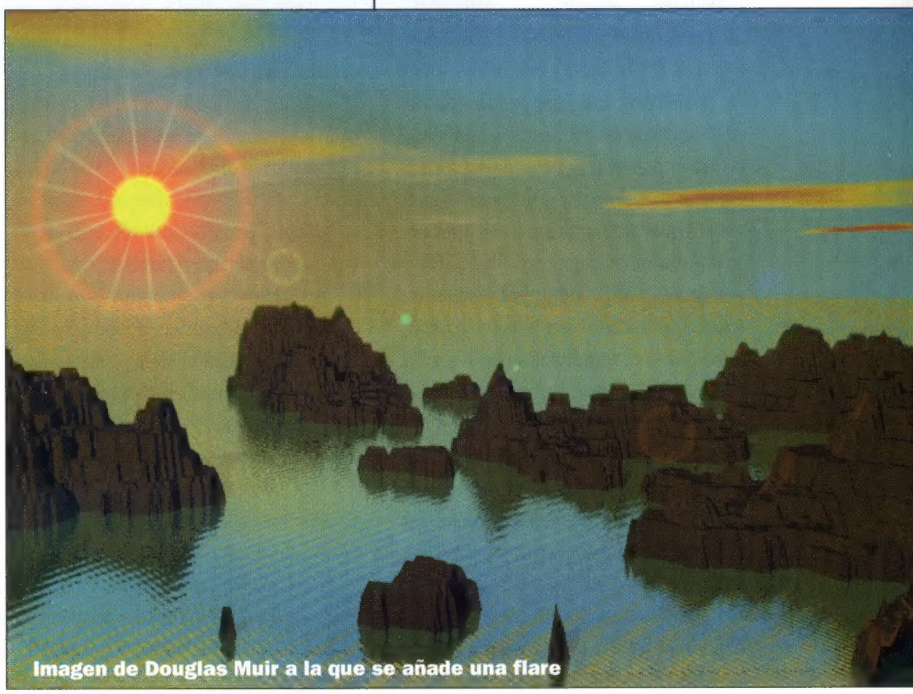
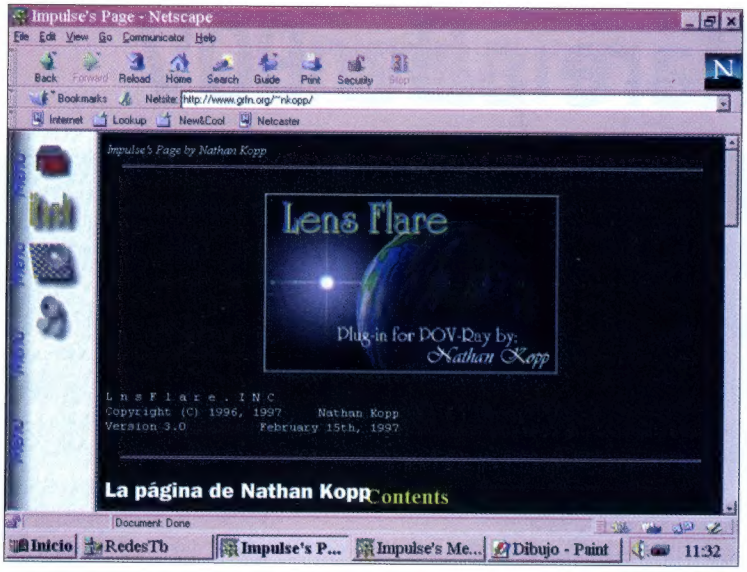


Imagen de Douglas Muir a la que se añade una flare



la imagen. Las flares de Polyray se añadían una vez que la imagen estaba calculada pero LnsFlare no necesita hacer esto).

Podemos comprobar esto realizando otro experimento: en tierra1.pov se ha añadido al planeta Tierra en el centro de la imagen. La Tierra está mucho más cerca de la cámara que la fuente de luz y debería, por tanto, ocultarla. Pero no ocurrirá así y seguiremos viendo el flare, como antes. Esto, naturalmente, se debe a que estamos viendo un efecto cuya posición se ha determinado a partir de la dirección de la fuente de luz a la que hemos atado dicho efecto. No estamos, insistimos, viendo una fuente de luz sino un efecto.

Desde luego esto puede resultar fastidioso en ciertos casos puesto que puede interesarnos que el espectador suponga que el flare es, como parecía al principio, una fuente estelar. Supongamos que estamos realizando una animación en la que precisamente hay un sol y un planeta. Tendríamos que tener cuidado de que el planeta no se interpusiera entre el sol y la cámara, ya que entonces quedaría claro que el flare no es más que un efecto. Pero, por suerte, Kopp pensó en esto ya en la versión 2.0 e implementó la posibilidad de ocultar el flare. Veamos cómo se hace esto: ante todo hay que decirle a LnsFlare dónde está el mundo que va a ocultar al flare y el radio del planeta:

```
#declare planet_loc = <X,Y,Z>
#declare planet_rad = Rmundo
```

El vector indicado en planet_loc tiene la posición del planeta y "Rmundo" es el radio del mismo. También habremos de indicar el radio de la fuente asociada al flare con...

```
#declare light_rad = Rluz
```

Si hecho esto renderizamos de nuevo veremos que POV imprime un mensaje diciendo que la fuente de luz queda detrás de la cámara o detrás de un planeta, y que el flare no se ha creado. Sin embargo, si alteramos el valor de la variable light_loc de tal modo que la fuente no quede oculta por el planeta aunque sí cerca de algún punto de su perfil, entonces POV imprimirá un mensaje diciendo que será visible un porcentaje del flare. De esto parece deducirse (aunque no hemos realizado la prueba) que podríamos realizar una estupenda animación en la que un sol apareciera de detrás de un planeta.

Kopp también ha previsto el poder hacer lo mismo con un plano, de manera que podamos representar una puesta de sol. Para ello emplearemos las variables siguientes;

```
#declare plane_loc  = <x,y,z>
#declare plane_norm = <x,y,z>
```

"Plane_loc" guarda las coordenadas del plano y "plane_norm" su normal (Nota: por ahora solamente puede especificarse un planeta o un plano que oculten al flare).

Manipular flares

Los flares de este "ipas" están compuestos por partes cuyas características podemos alterar cambiando los valores dados por defecto. En primer lugar tenemos el flare inicial o primario. Éste consiste en un brillo central con el que el espectador identificará a la fuente. Luego, extendiéndose a partir de este flare inicial, tenemos una zona semitransparente (glow) cuyo resplandor va decreciendo a medida que nos alejamos del flare primario. Esta zona puede extenderse incluso más allá del anillo. Después tenemos un anillo (ring) que engloba al flare inicial (y normalmente también al glow). Además, nor-

malmente suele haber unos destellos (sparkles) que parten del flare primario y, opcionalmente, unas bandas (streaks) que atraviesan el flare inicial (y que suelen asociarse a las explosiones estelares o a los brillos de los focos).

Aquí acaba lo que podemos considerar como la parte principal del flare. Luego siguen otros flares secundarios a los que llamaremos flare spots y flare dots. Ambos se dibujan formando una línea que cruza el flare inicial (en light_loc) y el punto central de la imagen (definido en lookat). Los flare spots son los empleados por defecto. Tienden a ser más transparentes en el interior que en sus bordes y por ello son llamados a veces ring spots. En los otros sucede exactamente a la inversa y también se distinguen por ser (por defecto) más pequeños que sus hermanos.

Estallido de un supernova

Veamos ahora cómo funciona todo esto. Kopp ha incluido en el archivo la definición de una veintena de flares que nos servirán para muchos casos diferentes. Ya hemos visto cómo usar estas flares predefinidas pero, en la práctica, es casi seguro que el usuario intentará crear enseguida sus propias flares empezando, quizá, por especificar un tipo concreto para realizar experimentos sobre él. ¡Craso error! La sorpresa del pov-adepto resultará mayúscula al principio, cuando se percate de que sus cambios en las variables de tamaño, color y otras, no tienen el menor efecto. No obstante, la explicación es muy sencilla: Kopp no ha empleado las sentencias #ifndef de la manera que podríamos suponer, colocando #ifndefs en todas las variables de cada tipo de flare. De haberlo hecho así podríamos declarar las variables del flare, y luego invocar al tipo deseado con la seguridad de que los cambios tendrían efecto. Esto sería lo más fácil pero podemos olvidarnos de ello. La única manera de conseguir cambios en un flare predefinido es, pues, cambiar las variables correspondientes en LnsFlare.inc.

Pero hay una excepción a esto. Si eliminamos la declaración de la variable "flare_type", LnsFlare entenderá que queremos emplear el flare por defecto, el cual sí utiliza #ifndefs en todas sus variables. La solución pasará pues por no emplear flare_type y copiar los declares del flare sobre el que pretendamos experimentar. A menos, claro está, que pretendamos empezar desde cero o alterar al propio flare por defecto.

Variables de control del flare primario

Sabido esto ya estamos en condiciones de empezar a estudiar las variables de LnsFlare.inc. En primer lugar tenemos las que afectan a las dimensiones del Flare. Estas son source_size, glow_size, ring_size y ring_width. La primera afecta a las dimensiones del flare primario y su valor es, por defecto, de 0.02. Las siguientes son, respectivamente, las dimensiones del glow, del anillo y del ancho del anillo. Es importante aclarar que estos valores son relativos al tamaño global de la pantalla y que es por esta razón por la que el flare no crece ni disminuye de tamaño cuando la cámara o la fuente se alejan. Sólo hay una excepción para esto: si cambiamos el valor "telescópico" de la

sentencia "direction" de la cámara (que se emplea para hacer zooms), entonces el tamaño del flare sí quedará afectado.

Ahora veamos las variables que controlan el color. Las de control directo son source_color, glow_color y ring_color. Como ya imaginará el lector, controlan respectivamente el color del flare primario, del glow y del anillo. Otras variables como source_bright y glow_bright son factores que se emplean para multiplicar el brillo del flare primario y su glow, controlando así su máximo nivel de brillo. Sin embargo, estas últimas variables no producirán efectos visibles si el color asignado ya estaba a su máximo valor de luminosidad. Otras variables que afectan al flare son las que controlan el nivel de translucencia. Estas son source_tr, glow_tr y ring_tr que afectan respectivamente al nivel mínimo de translucencia del flare primario, del glow y del anillo.

Destellos (Sparkles)

Los destellos son brazos radiales que parten del flare primario. Los hay de dos grupos: en el primero los brazos son más largos que los del segundo grupo, siendo los destellos de este último más numerosos que los del primero. La longitud de estos brazos se define en las variables star_size (primer grupo) y star2_size (segundo grupo). El número de destellos se indicará en num_arms y num_arms2, el ancho de los brazos se especificará en star_width y star2_width y su color en star_color. Otras variables importantes son star_tr, que controla el nivel mínimo de translucencia, star_seed, que guarda un valor de semilla para la aleatoriedad en la colocación de los destellos, y crisp_star, que causa el que los destellos del primer grupo tengan bordes definidos (ver el flare "concert"). Por supuesto podemos eliminar los destellos poniendo num_arms y num_arms2 a cero.

El planeta tapa parcialmente al flare

Flare spots y dots

Las características de los flares secundarios son controladas mediante una serie de variables que describiremos a continuación: Spot_size controlará el tamaño standard que se aplicará para estos flares y spot_color guardará su color normal (luego veremos el porqué de la palabra "normal"). Spot_tr guardará la translucencia mínima del spot y num_spots indicará al "ipas" el número de flares secundarios que se dibujarán a cada lado del flare primario. La separación entre los flares dependerá en principio de la distancia entre el flare primario y la posición observada pero podemos influir sobre dicha separación con un par de variables; spot_dist y spot_dist_pwr. La primera es un factor pero, en el momento de escribir estas líneas, no comprendemos aún su funcionamiento exacto. En cuanto a la segunda, determina si el crecimiento de las distancias de los flares secundarios con respecto al primero es lineal (valor 1) o cuadrático (valor 2). Otra variable interesante es skip_spots con la que indicaremos el número de spots que pueden omitirse (y que serán los más cercanos al flare primario).

Ahora vamos con un detalle muy interesante. Las variables descritas hasta ahora afectarán por igual a todos los spots con lo que el resultado final será excesivamente uniforme. Para remediar esto y dar algo de alegría al conjunto disponemos de algunas variables

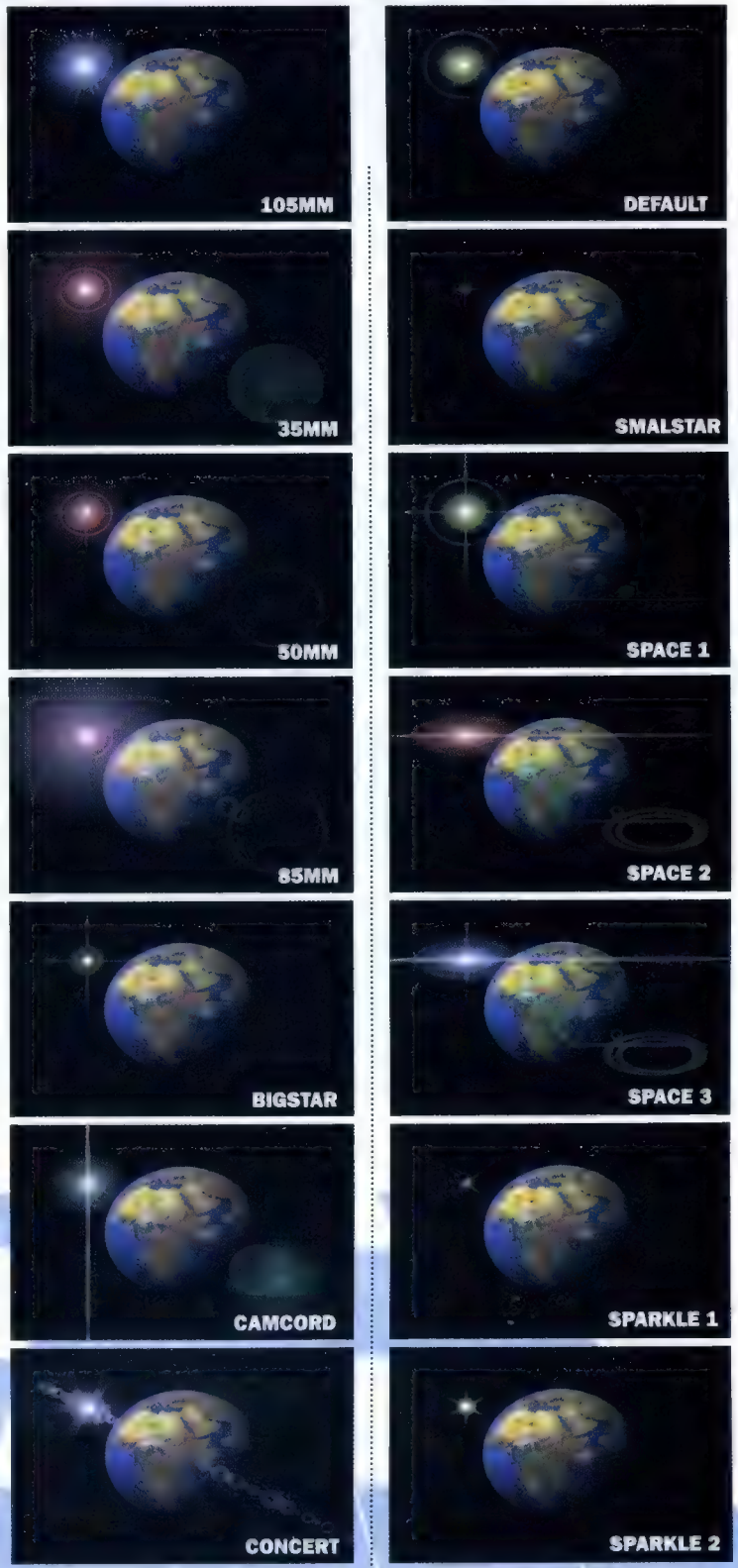
que introducirán un factor de aleatoriedad en los flares secundarios. Estas variables son spot_size_rand, spot_color_rand, spot_dist_rand y spot_seed. Los valores (de 0 a 1) que demos a estas variables se entenderán como un porcentaje de desviación con respecto a los valores "normales" definidos por las variables de tamaño, color y distancia que ya hemos visto. Por ejemplo, un valor de 0.2 en spot_size_rand implicará que el tamaño de los flares secundarios oscilará entre un 80% y un 120% con respecto al definido en spot_size. Spot_seed es, por supuesto, la semilla con la que alimentaremos al generador pseudoaleatorio.

¿Y cuándo se dibujan los flare dots? Bien, para eso está la variable llamada percent.dots, donde indicaremos un valor de 0 a 1 con el porcentaje de flares secundarios que deseamos que sean flares dot. Estos flares tienen su propio grupo de variables; dot_size, dot_color y dot_tr, cuyo significado ya habrá adivinado el lector.

¡Pero aún hay más! A partir de la versión 2.1, Kopp añadió la posibilidad de que los flares se visualizaran como hexágonos. Como anteriormente, para determinar el porcentaje de flares que deberán verse así, existe una variable determinada; percent_hex. Las características de estos flares serán controladas con las variables que empiezan con la palabra spot.

Bandas de luz (flare streaks)

Las bandas de luz son una característica que se encuentra desactivada por defecto. Estas bandas suelen ser horizontales o verticales y pueden emplearse para producir la imagen de una explosión estelar (ver flare space2) o



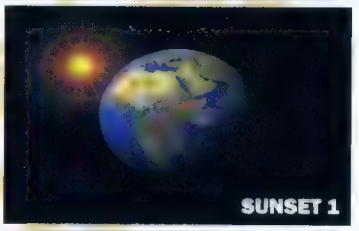
del foco típico de un campo de fútbol (flare sports). Las variables que controlan su funcionamiento son: streak_a_size, que controla el radio del flare, streak_a_color, con el que se especifica el color del borde de la banda, streak_a_center_color, que indicará el color del centro de la banda, streak_tr, que se usa para indicar la translucencia mínima de la banda, streak_a_scale, en el que guardaremos un vector para indicar la escala de la banda en los ejes X e Y, y streak_a_rotate, que se empleará para rotar la banda.

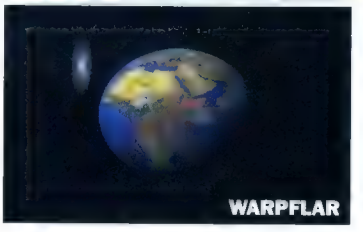
El lector habrá observado el empleo de una 'a' en todas estas variables. Esto quiere decir que dichas órdenes se limitan a la banda 'A' y que existe una segunda banda llamada 'B' para la que hay otro juego de variables que emplean la 'b'. Esto podemos comprobarlo examinando las líneas del flare sports, en el que se emplean las dos bandas y donde éstas se giran 45 grados.

Variables de uso general

El tamaño global de cualquier flare entero puede controlarse con la variable flare_scale_factor, cuyo valor por defecto es de 1. Si ponemos un valor de 2 en esta variable el tamaño del flare se doblará y si colocamos un valor 0.5, se dividirá por 2. Flare_rotate se emplea para rotar el flare. Flare_brightness (que guarda valores normalmente superiores a 1) se utiliza para ajustar el brillo global del flare y suele usarse cuando el flare ha quedado demasiado oscuro para la imagen.

Main_flare_scale se utiliza para controlar la escala del flare primario y de sus brillos y glow. Además podemos hacer que la escala a ordenar no sea idéntica en los dos ejes, con lo cual conseguiremos un interesante efecto de achatamiento similar al logrado en el flare space2.





Notas y trucos

Lo explicado hasta aquí debería ser suficiente para que los povmaníacos hagan sus propias escenas con este magnífico "ipas". Únicamente hay que recordar un par de cosas:

1) Algunas variables precisan valores que oscilan entre 0 y 1 y otras no. Si lo que hacéis no funciona echad un vistazo a las flares predefinidas. Es la mejor manera de asegurarse del tipo de entrada que precisa cada variable.

2) Podemos colocar más de un flare en la escena. Para ello bastará con colocar otra sentencia con un nuevo valor para light_loc y, por supuesto, con colocar otro include para LnsFlare.

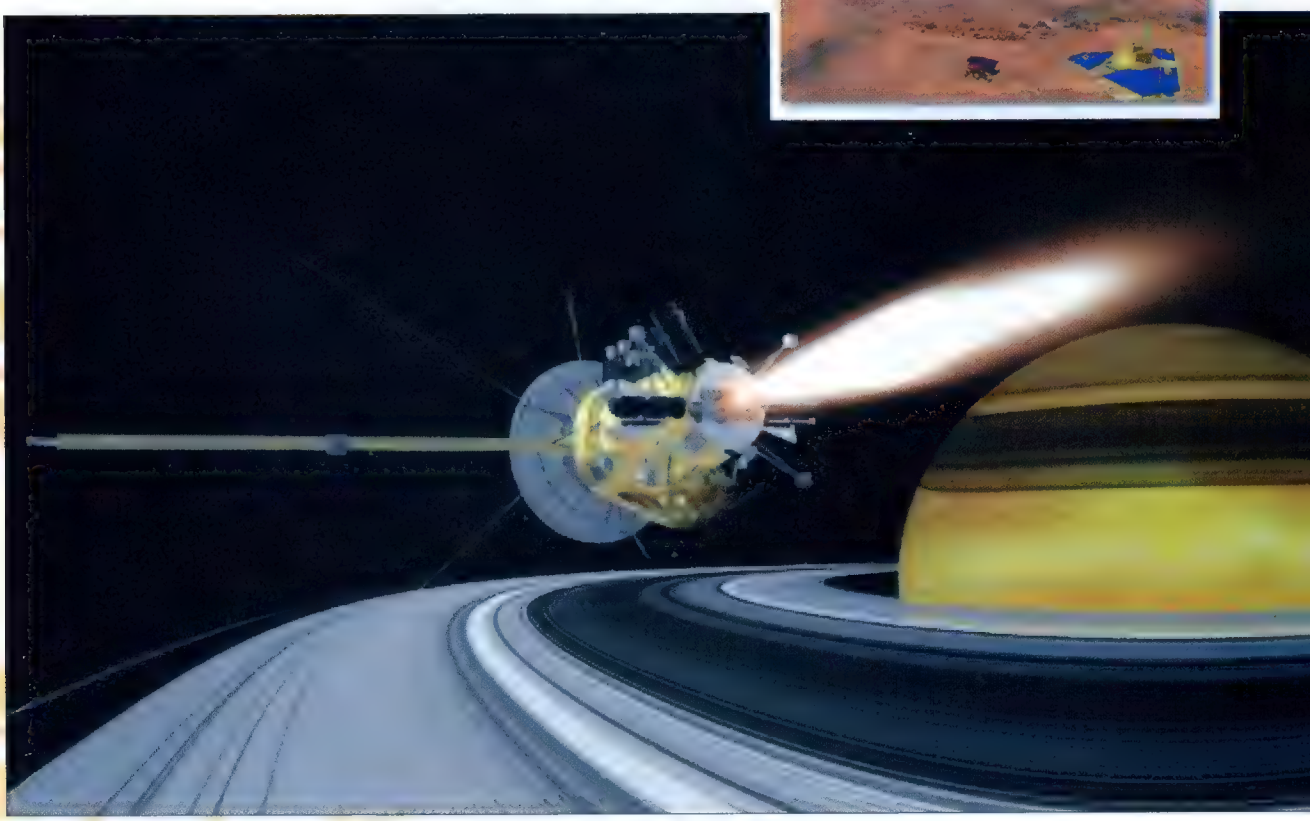
3) El flare que haya sido definido seguirá a la fuente a la que esté atado en una animación pero, en algunos casos, puede ser necesario emplear la sentencia "vrotate" del lenguaje escénico de POV.

4) Si hay problemas con espejos deberéis incrementar el valor de #max_trace_level.

5) Podéis encontrar próximas versiones de LnsFlare en el "site" de Nathan Kopp cuya visita os recomendamos. Está en la dirección http://www.grfn.org/~nkopp. En sus páginas, Kopp exhibe unas modificaciones de viejas obras de Douglas Muir y Dan Farmer.(hemos duplicado la de Muir añadiendo simplemente una flare, como hizo el mismo Kopp).

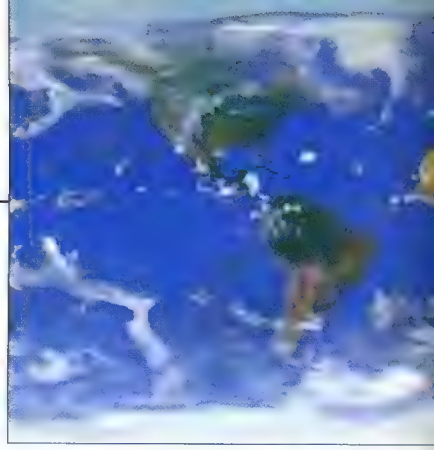
6) Los flares no funcionarán si se emplea algún tipo especial de cámara.

Arte Espacial



Este artículo se aparta un tanto de todo lo que es habitual en estas páginas pero creemos que por una vez valdrá la pena hacer una excepción. Todo empezó bastante inocentemente, buscando unos bitmaps planetarios por Internet, y acabó con un vistazo al arte espacial renderizado de la Nasa, una exploración marciana con MapMaker, y un recorrido por todo el sistema solar de la mano del simulador de la Jet Propulsion Laboratory.

Biblioteca 2D



Este artículo bien podría haber tenido un título diferente como por ejemplo; "Porque amo a Internet" o algo parecido. Sí, en efecto. Tenemos por fin una conexión propia y no podemos comprender cómo pudimos vivir antes sin ella. Sin duda ya estaréis hartos de oir frases como esta, pero la verdad es que llega un momento en que toda esa pesada palabrería quiere decir algo: rendermaníacos; la Red es algo maravilloso. Sólo el navegar, el explorar, ya es algo fascinante. Uno salta de un sitio a otro, la mayoría de las veces sin hallar nada útil, pero otras encontrando la perla que hace que todo valga la pena. Y además, la perla encontrada suele tener links que también tienen su interés.

Dónde encontrar bitmaps planetarios

El nuevo navegante ya no recuerda cómo encontró el Web de Thomas Constantine. El caso es que una de sus páginas, cuya dirección es...

www.lancs.ac.uk./postgrad/thomasc1/render/maps.htm

...pronto demostró ser muy interesante. Allí había un índice de mapas de planetas. Estaban la Tierra, Marte, Saturno, Urano, etc. Al pinchar en el link de "Earth", apareció una página con datos globales sobre nuestro mundo y también una ventana que exhibía un bitmap en proyección cilíndrica simple de la Tierra. Al pinchar sobre ella se cargó una preciosa imagen jpg con una vista de toda la corteza terrestre y de su atmósfera y ésta ha sido la textura que hemos empleado (en las pruebas de LnsFlare) para el planeta.Tierra. ¿Cómo? ¿Que hay alguien que no sabe lo que es un bitmap de proyección cilíndrica? Pues se trata de una textura que emplearemos para recubrir a objetos de forma cilíndrica o esférica. Las siguientes son las líneas con las que puede crearse a la "Tierra" desde POV:

```
sphere { pos_tierra, radio
  pigment {
    image_map{ tga "earth.tga"
map_type 1}
  }
  rotate y*-50
}
```

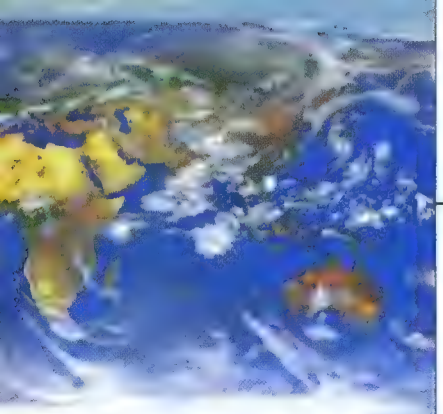
Desde luego la cosa es bien sencilla. Basta con crear una esfera y recubrirla con nuestro bitmap. Por si alguien aún no lo sabe, tendremos que emplear una proyección esférica (por defecto la proyección que realiza POV es plana y se extiende a lo largo del plano X-Y. Si queremos efectuar un mapeado esférico daremos el valor 1 a map_type). Usando la aplicación esférica, la textura indicada se mapeará adecuadamente sobre la esfera que hace las veces de planeta. El bitmap se enrollará sobre la esfera de manera que su borde superior quede en el polo norte y el inferior en el polo sur. La textura recubrirá completamente a la esfera una única vez sin que importe el tamaño que ésta pueda tener (la palabra "once" no funciona con este tipo de proyección).

Pero ahora volvamos con la navegación: después de retornar al índice, encontramos mapas similares que pueden ser usados para crear planetas con el aspecto de Marte, Júpiter, etc. El mapa de la Tierra es el producto resultante de miles de fotografías tomadas por una serie de satélites y por tanto es muy preciso. Otros, como el de Júpiter por ejemplo, han requerido bastante esfuerzo infográfico y artístico (empleando productos como Photoshop y otros) para conseguir el resultado final que podéis ver en el CD-ROM. En cualquier caso estos mapas –que podéis encontrar en el CD-ROM– pueden ser utilizados en simulaciones de viajes por el sistema solar o en escenas artísticas, y pueden ser empleados por cualquier programa 3D y no sólo por POV.

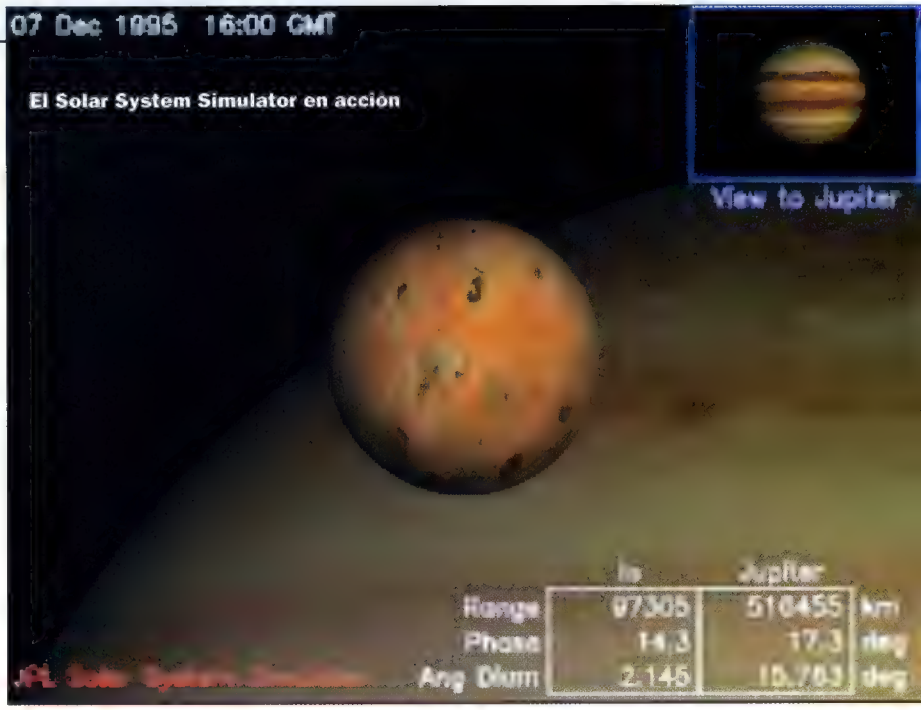
Arte planetario

Después de "bajarnos" unas cuantas texturas de este tipo, retornamos al principio de la página y leímos que muchos de los mapas que podían encontrarse antes en el Web de Thomas ya no

Una vista de Saturno desde uno de sus satélites



Bitmap para aplicación esférica



estaban, pero que había otras nuevas versiones en la página de un tal David Seal. Unas líneas más abajo se indicaba que David trabajaba en el Jet Propulsion Lab (JPL) y que era responsable de gran parte del trabajo artístico de la Nasa. ¿Trabajo artístico de la Nasa? Y se añadía que por esa razón los mapas de David eran mucho más grandes y detallados que los del propio Thomas...

Después de pinchar en el link de David se tomó nota de la dirección siguiente...

http://samadhi.jpl.nasa.gov/txmp

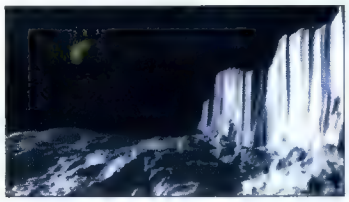
Allí, aparte de muchos más mapas, hay mucha información interesante. Eliminando el directorio txmp de la dirección accederemos a una página donde hay enlaces a muchos lugares de interés. Comenzaremos comentando la existencia del "Nasa Mission Artwork" (en http://samadhi.jpl.nasa.gov/art). Aquí pueden verse preciosas escenas renderizadas de sondas espaciales viajando por distintos puntos del sistema solar. En cada una de estas imágenes, se explica el punto de la misión que representa la escena, las maniobras que llevaron (o llevaran) allí a la nave, etc. Estas imágenes (o animaciones porque también las hay) fueron hechas con multitud de herramientas sobre plataformas SPARC y SGI (y parece ser que la más importante es SPACE; un conjunto de programas diseñados para simular trayectorias de naves espaciales y que la nasa emplea para crear simulaciones de misiones en fase de proyecto).

Otro lugar interesante es el "Solar System Surfaces", donde encontraremos renders de tipo artístico de diversas superficies de diferentes objetos del sistema solar (el sol, asteroides, planetas...) Algunas de estas escenas (que podéis admirar en el CD-ROM) son muy bellas. La dirección de esta página es...

http://samadhi.jpl.nasa.gov/surfaces

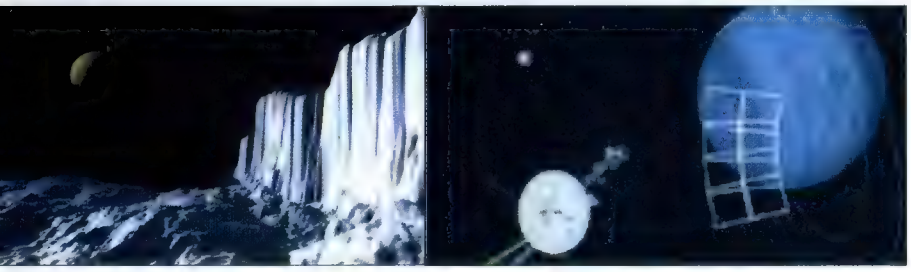
Pero continuenos con nuestro periplo. Otro enlace interesante de la página base es el "Spacecraft Models", en el directorio models, donde el infografista hallará una lista de modelos 3D de las sondas de la Nasa (el problema es que hay muy pocas en formato dxf).

Y finalmente, y ya para acabar con este sitio, tenemos el "Solar System Simulator". Parece ser que en junio del 97 David Seal terminó de adaptar el software de Space y lo conectó al World Wide Web. Este nuevo software viene a ser un completo simulador del sistema solar, ya que puede renderizar cualquier punto del mismo teniendo en cuenta la posición del observador, el punto hacia el que mira, la fecha, el campo de vista, etc. Las texturas empleadas en los planetas y otros cuerpos han sido obtenidas de diversas colecciones de fotos tomadas por sondas espaciales y muchas de ellas pueden ser encontradas en la página de mapas que hemos comentado anteriormente. Según se rumorea, este software podría estar disponible en un futuro próximo aunque, claro está, no lo cataremos los infografistas de a pie (¿alguien tiene aquí una SPARC?)

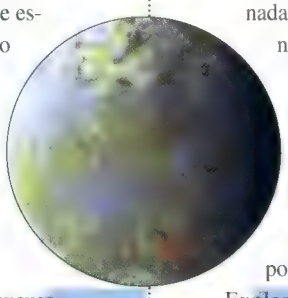


Viaje fotográfico por los planetas

Perdido entre las páginas anteriores hay un link de un lugar llamado "Nasa's Planetary Photojournal". Se trata de una enorme colección de fotografías tomadas por sondas espaciales y que los aficionados a la astronomía podrán bajarse tranquilamente, ya que han sido cedidas al uso público. Hay que preci-



sar que se trata de fotografías y no de imágenes retocadas y preparadas para ser empleadas infográficamente. El usuario podrá solicitar las fotos directamente (si conoce la codificación empleada) o podrá ir examinando las fotos de una sonda determinada, tras lo cual podrá bajarla a su modesto PC. Hay que advertir, sin embargo, que muchas de estas fotos tienen una resolución grande o muy grande y que el cibernauta tendrá que tener cuidado, so pena de perder el tiempo mirando una enorme foto de la luna u otro cuerpo durante más tiempo del que querríamos. Esta colección, que crece diariamente, tiene, en estos momentos de escribir estas líneas, algo menos de 700 fotos y su dirección es: **http://photojournal.jpl.nasa.gov**



Mapas marcianos a medida

Otro lugar a visitar inexcusablemente es el Charlotte´s Web Server, desde el que podremos acceder a muchos lugares de interés. Su dirección es "http://www-pdsimage.jpl.nasa.gov" y desde allí podremos saltar al "Solar System Visualization Proyect" (que resultó ser horriblemente lento en las pruebas) o al "Planetary Data System Imaging Node". Este último parece ser un sistema que mantiene y distribuye los archivos de fotos y datos de la Nasa, a fin de que la comunidad científica tenga un fácil acceso a ellos. A través del "Node" puede accederse a un sistema de ficheros almacenados en CD-ROM por el que los expertos podrán acceder a datos de misiones históricas que, según parece, estaban a punto de perderse al no haber antes un sitio especializado para ellas. Así pues muchas de las imágenes almacenadas (hay algunas excepciones) podrán ser accedidas a traves del software de extracción de datos del mismo Web, cuya dirección de Internet es:

http://www-pdsimage.jpl.nasa.gov/PDS

La cantidad de información almacenada aquí es enorme. Por poner un ejemplo, solamente las fotos accesibles bajo el epígrafe de las sondas Viking ya son 50.000. Y hablando de las sondas Viking, desde la dirección anterior podremos acceder al "Mars Explorer" un programa por el que el usuario podrá crear su propio mapa de la superficie marciana escogiendo su propio factor de zoom, tamaño del mapa y sistema de proyección. Los mapas se confeccionarán gracias a un software llamado MapMaker que trabajará sobre las fotos de las misiones Viking para crear el mapa que deseemos. Según parece las fotos están almacenadas en un sistema de CD-ROM y el programa nos permitirá acceder a ellas vía Internet de una manera sumamente atractiva: pinchando en un mapa global de Marte pincharemos èn la zona que nos interese. Entonces, tras una corta espera, MapMaker compondrá un mapa con las fotos de la zona y nos lo enviará. Podremos entonces seguir haciendo ampliaciones de la misma manera o bien desplazarnos por la superficie marciana con las flechas de movimiento. Naturalmente llegará un momento en que, si seguimos ampliando y ampliando, llegaremos al límite de resolución de las fotos y no será posible continuar con el zoom (el uso de MapMaker es muy divertido. Pensadlo: estamos viajando por Marte).

La comodidad que proporciona MapMaker al usuario es enorme (y, además, estos mapas sí pueden ser empleados en escenas sintéticas). Además, según se dice, es posible que en un futuro próximo MapMaker soporte a otros cuerpos celestes como la Luna o Venus.

Todo tiene un final

Durante este viaje por la Red hallamos texturas, fotos, animaciones, faqs, etc. El caudal de datos (y de siglas de organizaciones que los mantenían) no parecía tener fin. También encontramos otros "sites" de arte espacial hecho por usuarios (muchos de ellos de POV) e incluso naves espaciales para los más diversos programas (incluyendo a Imagine y POV) pero ya hablaremos de ello en otro momento.

En cuanto a los bitmaps planetarios, estos están en formato jpg por lo que tendréis que convertirlos a TGA con alguna utilidad como Alchemy o Photoshop antes de emplearlos para crear vuestros planetas (todos ellos empiezan con la letra "t"; Tearth, Tmars, etc).

El Foro del Lector

Nota importante. Podéis remitirnos vuestros trabajos o consultas, bien por carta a la dirección que figura en la segunda página de PCmanía, o vía e-mail a rendermania.pcmania@hobbypress.es

Hoy, 3D Studio

Este mes –y salvo excepciones– casi todos los trabajos recibidos tienen en común el haber sido creados con «3D Studio». Sin embargo, esta es la única coincidencia, ya que en todo lo demás la diversidad es la norma en estas páginas en las que podremos ver soldados medievales, Mechs de combate, animaciones varias y otros trabajos de interés.

Una de las mayores recompensas de nuestro trabajo consiste en leer las cartas de lectores que nos envían ánimos, elogios, críticas o, simplemente, el mensaje de que nuestros artículos han servido de algo. Misivas como la de **Julián Hierro García** –que podéis encontrar en el foro– caen en este apartado. Julián, ya un viejo conocido de Rendermanía, asegura haberse empollado la serie de artículos dedicados a Imagine y ahora recibimos el fruto de su trabajo: el temible battlemech de la serie 201, un mech de combate diseñado por Julián, completamente articulado y dispuesto para la lucha (que podréis encontrar en el CD-ROM). Julián nos pide una crítica de su mech pero la verdad es que ésta se nos hace muy difícil porque lo cierto es que nos ha gustado todo, incluso la textura. Los puntos de articulación para el 201 parecen correctamente dispuestos, la forma es interesante, el pie es muy majo y Julián demuestra en su carta un buen conocimiento de Imagine. Como puede verse en una de tus escenas, tu mech es mucho más pesado que el nuestro. ¿No resultará algo lento a pesar de su nuevo reactor Pullman 1.6?

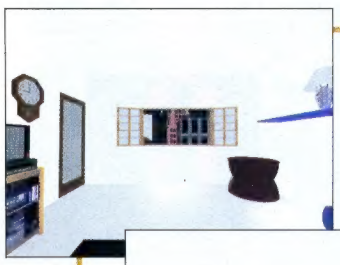
Julián también pide la opinión de otros rendermaníacos sobre su trabajo y, como el movimiento se demuestra andando, él hace lo propio con algunos viejos conocidos como Díaz, Sendin, Torres y otros. ¡Enhorabuena por tus Imagine-criaturas! (Naturalmente esperamos las próximas, ya sean de POV o Imagine).

El Foro del Lector

Felipe Camacho Sillero nos envía uno de sus primeros trabajos realizado con «3D Studio». Se trata de un modelo con el escudo de su equipo de fútbol favorito; el Córdoba. Felipe incluye además una sugerencia en su carta solicitando un curso de «3D Studio». Bien, por el momento no hay nada previsto sobre ello pero nunca hay que descartar ninguna posibilidad...

Antonio Abenza Frau nos envía también sus primeros trabajos hechos con el programa «3D Studio 4». Algunos de ellos –como el taller– tienen más merito y trabajo del que parece (el cable del soldador, las tenazas, etc) y por ello te enviamos la oportuna felicitación. Tan sólo te diremos que deberías haber enviado una malla o un texto explicativo que ayudara a certificar la autoría de tus imágenes.


Imagen creada por Antonio Abenza.



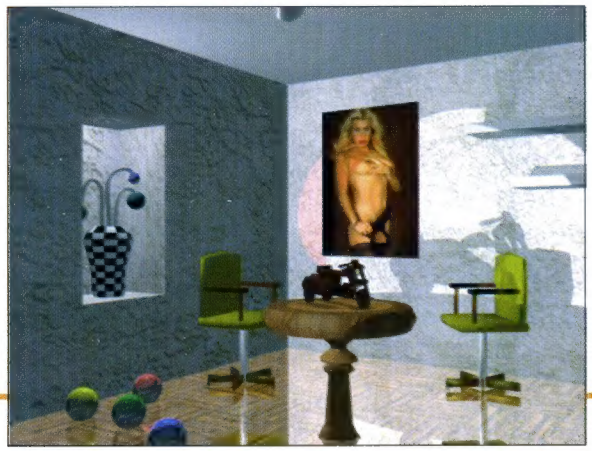
La primera animación de **Carles Vila** era un recorrido casi completo por una casa. Desdichadamente dicha animación ocupaba unos 62 Megas y por esa razón Carles ha debido contentarse con enviarnos otra que sólo tiene unos 2 Megas. (Por cierto, nuestro más sentido pésame por los cuelgues que se producen en tu ordenador). Esta animación está realizada con «3D Studio 4».

José Miguel Domínguez Montesinos es otro rendermaníaco que también envía sus primeros trabajos con «3D Studio 4». José Miguel no ha enviado tampoco ni mallas ni excesivos datos acerca del proceso de creación de sus trabajos y por ello le enviamos el oportuno tirón de orejas. José Miguel apunta en su carta algunas preguntillas:

1) ¿Cómo hacer que las superficies de «3D Studio» no queden tan perfectas? Como siempre hay diversos caminos pero casi todos pasan por empollarse bien el Materials editor. Lo más sencillo es aplicar algún bitmap con apariencia de metal corroído y eliminar los brillos de las superficies. Otro sistema menos utilizado es emplear algún Ipas de fractalización que desordene al azar (en un pequeño factor) los vértices.

2) ¿Dónde pueden encontrarse en Internet Ipas para «3D Studio»? Esto no es fácil ya que la inmensa mayoría de los Ipas son de pago y no se distribuyen libremente (salvo en versiones recortadas). Prueba a mirar cada cierto tiempo en el Web de Avalon, donde también podrás hallar objetos, texturas, informaciones diversas, etc. (En http://avalon.viewpoint.com o a través de www.viewpoint.com). Otro lugar interesante es el 3dcafe en www.3dcafe.com

3) ¿Que si se pueden conseguir imágenes de buena calidad con POV? Dios mío...

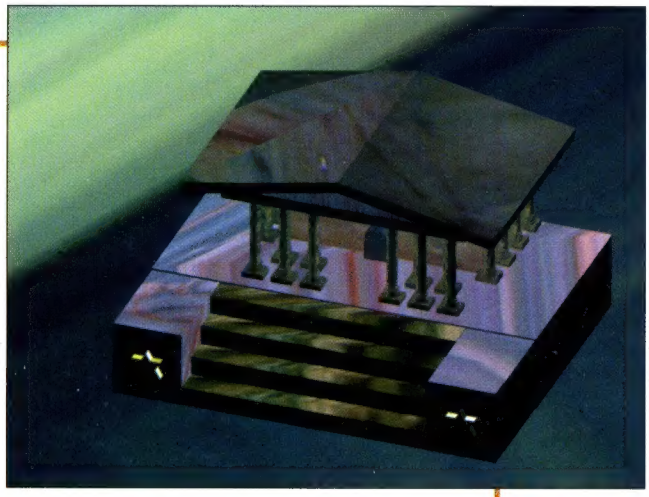


Héctor de la Fuente Pita nos envía, entre otros modelos, un soldado medieval creado con «3D Studio 3». Según afirma Héctor, casi todas las piezas del modelo –cuya malla podéis encontrar en el CD-ROM– fueron construidas con la opción Fit a partir de formas creadas desde el 2d Shaper. Para este modelo Héctor solicita una ¿severa? crítica. Bueno, las proporciones generales del soldado son correctas y éste merece todo el reconocimiento que debe darse a cualquier modelo antropomórfico aceptablemente resultón. Quizá hubieras debido añadirle algunas piezas sueltas (un faldellín, un peto, etc.) que hicieran las funciones de armadura ligera y haber aplicado al modelo actual una textura que recordara a las típicas cotas de malla de la época. Las manos tampoco son muy buenas pero es dificilísimo hacer unas manos aceptables con técnicas normales.

El soldado llegó con algo de retraso para la batalla (que de todos modos se quedó en escaramuza por falta de memoria) pero es posible que cuando haya orcos, este modelo pueda ser aprovechado (al igual que otros).

En cuanto a tu sugerencia, todos los lectores de PCmanía pueden formar sus propias colecciones de objetos a partir del material que se va publicando aquí todos los meses. Lo único que queda es sugerir temas, lo cual ya está previsto para el asunto de las portadas. Por lo demás la organización de los modelos según los temas podría ser algo problemática ya que requeriría un tiempo del que ahora carecemos (y además habría que escribir sobre cosas que ya se habrían comentado en números previos, lo cual no nos parece un ahorro de espacio).

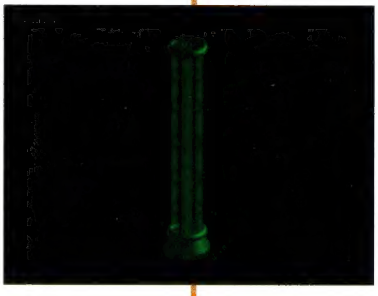
El Foro del Lector



Pedro Palazón Martínez nos envía algunas imágenes creadas con «3D Studio 4» con las que pasa a formar parte de la cada vez más extensa lista de rendermaníacos infográficamente activos. Pedro solicita alguna guía para mejorar, y desde aquí te damos un consejo muy sencillo: practica mucho el modelado y no emplees tantísimos polígonos para algo que puede representarse igual con mallas mucho menos densas (basta con ver la columna del templo). En cuanto a tus sugerencias...

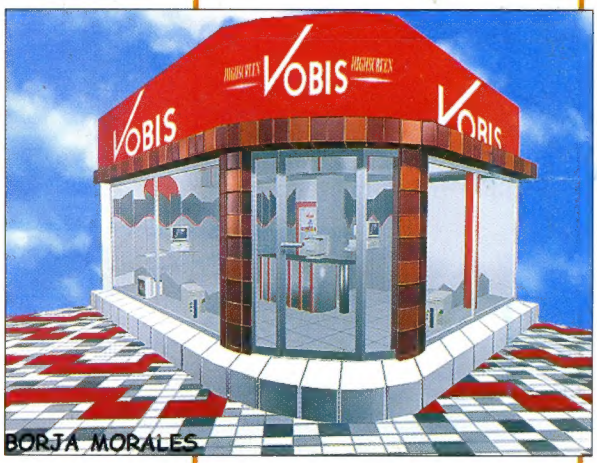
1) Lo de incluir texturas de libre distribución es una buena idea pero no hay tantas (de calidad) como imaginas. ¿Qué te parecen las texturas planetarias que hemos enseñado en el presente número?

2) En cuanto al debate, sería muy difícil evitar que acabara reducido a una mera publicación de opiniones (¡no puede haber tiempo real!). En algún momento han surgido temas que provocaban el envío de opiniones por parte de los lectores (para eso está también el foro) pero no creemos conveniente provocar debates porque sí. Lo ideal para cualquier rendermaníaco es practicar, crear cosas, e intercambiar conocimientos relacionados con cosas concretas.



En cuanto a tus preguntas, la utilidad 3ds2pov puede convertir ficheros de 3D Studio a POV con casi total perfección, ya que se traducen modelos, colores, posición de las luces y de la cámara, etc. La imagen generada en POV será prácticamente idéntica a la creada por 3D Studio salvo por lo referente a los bitmaps, si los hay. Lo contrario (POV a 3D Studio) no es posible en la inmensa mayoría de los casos debido a que las primitivas de POV no se construyen con polígonos. Y para cuestiones arquitectónicas preferimos POV a 3D Studio (tu otra pregunta), aunque esto siempre es algo muy subjetivo.

Borja Morales alias «3d reality» ha enviado varias animaciones creadas con «3D Studio 4». Pero, desgraciadamente, uno de los ficheros del paquete (el reality.a03) llegó estropeado y sólo fue posible salvar a mago.avi. En esta animación, Borja ha creado unas secuencias de gran realismo empleando el magnífico mago creado por Juan Carlos Jiménez Méndez. En la película el mago corre, se detiene, prepara una bola de energía mágica y la lanza para, finalmente, ejecutar un saludo ritual. Los resultados son sencillamente magníficos y en ellos se aprecia que, salvo en unos minúsculos detalles, el movimiento del personaje es perfecto. El mago no sufre prácticamente deslizamientos al correr (fijaos en la toma lateral) y pueden apreciarse bien los cambios de altura debidos a este ejercicio. Lo único que desentona (en cuanto a nivel de realismo) son los movimientos de los pies del personaje cuando éste ya se ha detenido y está preparando y lanzando la bola mágica (porque se producen unos deslizamientos bastante raros). Pero, salvo este pequeño detalle, la animación es extraordinaria. ¡Enhorabuena amigo Borja!



Por varias razones tenemos que enviar un tirón de orejas a nuestro amigo **Oriol Bagan.** Este lector nos ha enviado su primer trabajo (con «Imagine 3.0») basado en un juego llamado Ballz. Pero, Oriol no ha incluido ni un solo render del modelo, razón por la cual no hay aquí ninguna foto del mismo. Además, tampoco diste la extensión adecuada (obj) al archivo, con lo cual al principio lo tomamos por una imagen en algún formato raro.